Zookeeper service

Consists of individual servers

Leader

One server in the service is the leader

All servers are aware of the leader

Data storage

Designed to store small pieces of data

Follower

Non leader in a zookeeper cluster

Client operations

Clients connect to a single zookeeper server

Read requests are serviced by the server receiving the request

Write / update requests follow a special protocol

hierarchical namespace

Client protocols

Clients connect to a single zookeeper server over TCP

Clients may interact with zookeeper as follows

* send read requests
* send write requests
* register watches on znodes
* send heartbeats

Agreement protocol

Protocol that manages state updates

Any update request from a client is forwarded to the leader

Local replicas

Replicated state of zookeeper service present on all zookeeper servers

Guaranteed to be consistent always as a result of the underlying atomic messaging protocol

Atomic messaging protocol

Ensure that local replicas are always consistent

Zookeeper session

Created when a client connects to a server

Represented by a 64 bit number

Clients may create ephemeral znodes while part of a session.

An ephemeral znode’s ephemeral owner is the sessionid of the client that created the node

Versioning

ZNodes contain three version numbers

version

number of changes to the znode data

cversion

number of changes to children of a znode

aversion

number of changes to znode

State

State is stored in data registers called znodes

A zookeeper tree may contain many znodes

ZNodes have a hierarchical tree-like structure

A ZNode always has a single parent

Data is stored in memory

Replicated in-memory database

Enables low-latency and high throughput

Each server contains a copy of the entire data tree

All data is backed by a log to ensure recoverability

Any writes are applied to disk before being applied to memory

ZNodes are the elements on which Zookeeper trees are created

Clients connect to zookeeper by connecting to a quorum peer

ZNodes store state

State is written to / read from a znode is done so atomically

Any read obtains all the data stored in a znode atomically

Any write overwrites all the data sored in a znode atomically

Access to any individual znode is managed by that node’s ACL

Every znode is accessible by an absolute path

Each update is stamped with a unique incrementing id

Clients

Clients connect to an individual server in the ensemble via TCP connection

Clients interact with the server as follows:

* send requests
* get responses
* get watch events
* send heartbeats

If client gets disconnected from the server, it will reconnect to a different server

API

Reference

http://www-01.ibm.com/software/data/infosphere/hadoop/zookeeper/